2018

AP CollegeBoard

AP Computer Science A

Sample Student Reponses and Scoring Notes

DRAFT

These exam materials may ***not*** be posted on school or personal websites, nor electronically redistributed for any reason without the express permission of the College Board.

These training materials may be used solely for noncommercial purposes by AP teachers and students for course exam preparation. Permission for any other use must be sought from the College Board. Teachers may reproduce these training materials in whole or in part, in limited quantities, for noncommercial, face-to-face teaching purposes, as long as the copyright notices are kept intact.

© 2018 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board. Visit the College Board on the Web: www.collegeboard.org.

AP Central is the official online home for the AP Program: apcentral.collegeboard.org

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
   /** Distance, in inches, from the starting position to the goal. */
   private int goalDistance;

   /** Maximum number of hops allowed to reach the goal. */
   private int maxHops;


   /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
    *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
    *  Precondition: dist > 0; numHops > 0
    */
   public FrogSimulation(int dist, int numHops)
   {
      goalDistance = dist;
      maxHops = numHops;
   }


   /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
    */
   private int hopDistance()
   {  /* implementation not shown */  }


   /** Simulates a frog attempting to reach the goal as described in part (a).
    *  Returns true if the frog successfully reached or passed the goal during the simulation;
    *          false otherwise.
    */
   public boolean simulate()
   {  /* to be implemented in part (a) */  }


   /** Runs num simulations and returns the proportion of simulations in which the frog
    *  successfully reached or passed the goal.
    *  Precondition: num > 0
    */
   public double runSimulations(int num)
   {  /* to be implemented in part (b) */  }
}
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	`24`	`true`
Example 2	`6, 7, 6, 6`	`25`	`true`
Example 3	`6, -6, 31`	`31`	`true`
Example 4	`4, 2, -8`	`-2`	`false`
Example 5	`5, 4, 2, 4, 3`	`18`	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.2

Question 1: Frog Simulation

Part (a)

```
public boolean simulate()
{
   int position = 0;

   for (int count = 0; count < maxHops; count++)
   {
      position += hopDistance();
      if (position >= goalDistance)
      {
         return true;
      }
      else if (position < 0)
      {
         return false;
      }
   }
   return false;
}
```

Part (b)

```
public double runSimulations(int num)
{
   int countSuccess = 0;

   for (int count = 0; count < num; count++)
   {
      if(simulate())
      {
         countSuccess++;
      }
   }
   return (double)countSuccess / num;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.2

Question 1: Frog Simulation

Part (a)	`simulate`	**5 points**

Intent: *Simulate the distance traveled by a hopping frog*

+1 Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position

+1 Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)

+1 Determines if a distance representing multiple hops is at least `goalDistance`

+1 Determines if a distance representing multiple hops is less than starting position

+1 Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position

Part (b)	`runSimulations`	**4 points**

Intent: *Determine the proportion of successful frog hopping simulations*

+1 Calls `simulate` the specified number of times (*no bounds errors*)

+1 Initializes and accumulates a count of `true` results

+1 Calculates proportion of successful simulations using `double` arithmetic

+1 Returns calculated value

Question-Specific Penalties

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.2

Question 1: Scoring Notes

Part (a) `simulate`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position.	• use `hopDistance()` as a position, like `hopDistance() < 0`	• only use `hopDistance()` as a count, like `hopDistance() < maxHops`
+1	Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)		• do not use a loop
+1	Determines if a distance representing multiple hops is at least `goalDistance`	• use some number of hops `*` `hopDistance()` as the frog's final position	
+1	Determines if a distance representing multiple hops is less than starting position		
+1	Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position	• have checks for all three conditions and correct return logic based on those checks, even if a check did not earn a point	• do not check all three conditions • the only check for `goalDistance` is after the loop • the only check for starting position is after the loop
Part (b) `runSimulations`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `simulate` the specified number of times (*no bounds errors*)	• do not use the result of calling `simulate`	• do not use a loop
+1	Initializes and accumulates a count of `true` results		• initialize the count inside a loop • do not use a loop
+1	Calculates proportion of successful simulations using `double` arithmetic	• perform the correct calculation on an accumulated value, even if there was an error in the accumulation	• fail to divide by the parameter
+1	Returns calculated value		• calculate values using non-numeric types • return a count of simulations

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators **(× • ÷ ≤ ≥ <> ≠)**
- `[]` vs. `()` vs. `<>`
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "*ArayList*" instead of "*ArrayList*". As a counter example, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

1Aa

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate(){

int distance = 0;

for(int i=0; i< maxHops; i++){
   distance += hopDistance();
   if (distance >= goalDistance)
      return true;
   if (distance < 0)
      return false;
}

return false; // only occurs if the frog does not reach its goal
              // despite taking the maximum number of hops
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

/16

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
  double sum = 0;
  boolean result;

  for(int i = 0; i < num; i++)
  {
    result = simulate();
    if (result == true)
      sum++;
  }

  double proportion = sum / num;
  return proportion;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

1Ba

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
    int totalDistance = 0;

    for (int a=0; a<maxHops; a++)
    {
        totalDistance += hopDistance();
        if (totalDistance >= goalDistance)
        {
            return true;
        }
        if (totalDistance < 0)
        {
            return false;
        }
    }
    return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
    int passed=0;
    for(int a=0; a<num; a++)
    {
        if ( simulate() )
        {
            passed++;
        }
    }
    return (double)(passed/num);
}
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

/Ca

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
    int sum = 0;

    for(int x = 0; x < maxHops; x++) {
        sum += hopDistance();
        if (sum < 0)
            return false;
    }
    if( sum >= goalDistance)
        return true;
    else
        return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`. /C6

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
  double t = 0;
  double amt = 0.0;

  for(int x = 0; x < num; x++){
      simulate();
      if( simulate() == true)
          t += 1.0;
  }
  amt = t / (double)(num);

  return amt;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit. 1 Da

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
  int distance = 0;
  for(int i=0; i < Maxhops; i++)
      distance += hopDistance();

  }
  if(distance >= goalDistance)
      return True;

  }
  return False;
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
    double passed = 0.0;
    for(int i=0; i<num; i++){
        simulate();
        if(simulate() == True){
            passed++;
        }
    }
    return passed / num;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

1E a

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate(){
    int pos = 0;
    int hops = 0;
    while (hops <= maxHops && pos < goalDistance){
        pos += hopDistance();
        hops ++;
    }
    if (pos >= goalDistance)
        return true;
    else
        return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num){
    int x=0;  int y= 0;
    while(x < num){
        if(simulate())
            y++;
    }
    return (double) y/num;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

1Fa

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
  int position = 0;                    // starting position
  position += hopDistance;             // final position of frog
  if ( position >= dist)
  {
    return true;
  }
  return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
  int count = 0;
  for (int i = 0; i <= num; i++)
  {
    if (i.getSimulate() == true)
    {
      count++;
    }
  }
  return (double) count / num;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

16a

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
    int distance = 0;
    for(i = 0; i < maxHops; i++)
    {
        distance = distance + simHopDistance();
    }
    if (distance >= goalDistance)
    {
        return true;
    }
    else
        return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
    double success = 0;

    for (i = 0; i = int; i++)
    {
        if (Sum.simulate() = true)
            success++;
    }

    return success / num;

}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit. 14a

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
{
    int numHops = 0;
    int frogPos = 0;
    if((frogPos < goalDistance)||frogPos<0 || numHops>maxHops)
    {
      frogPos += hopDistance();
      numHops++;
    }
    if(frogPos > goalDistance)
    {
      return true;
    }
    return false;
}
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
{
  double numWins;
  for(i=0; i<=num; i++)
  {
    boolean reachGoal = simulate();
    if(reachGoal)
    {
      numWins++;
    }
  }
  return num/numWins;
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

I a

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 *  Returns true if the frog successfully reached or passed the goal during the simulation;
 *          false otherwise.
 */
public boolean simulate()
    counter=0;
    for (int i=0; i < goalDistance; i++){
        if (i.hopDistance < 0 || i.hopDistance > goalDistance || i.hopDistance > maxHops) {
            return false;
        } else {
            counter++;
            return true;
        }
    }
```

Part (b) begins on page 8

Unauthorized copying or reuse of any part of this page is illegal.

(b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is `400`, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return `0.25`.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
```

```
int successes = counter;
double prop = counter/num;
return prop;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Question 1

Sample Identifier: 1A
Score: 9

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☑ Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)
3: ☑ Determines if a distance representing multiple hops is at least `goalDistance`
4: ☑ Determines if a distance representing multiple hops is less than starting position
5: ☑ Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☑ Calls `simulate` the specified number of times (*no bounds errors*)
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

Sample Identifier: 1B
Score: 8

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☑ Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)
3: ☑ Determines if a distance representing multiple hops is at least `goalDistance`
4: ☑ Determines if a distance representing multiple hops is less than starting position
5: ☑ Returns `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☑ Calls `simulate` the specified number of times (*no bounds errors*)
7: ☑ Initializes and accumulates a count of `true` results
8: ☐ Calculates proportion of successful simulations but does not use correct `double` arithmetic
9: ☑ Returns calculated value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 1C
Score: 7

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☑ Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)
3: ☑ Determines if a distance representing multiple hops is at least `goalDistance`
4: ☑ Determines if a distance representing multiple hops is less than starting position
5: ☐ Returns `false` if goal never reached or position ever less than starting position, but does not immediately return `true` if goal reached
6: ☐ Calls `simulate` more than the specified number of times
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

Sample Identifier: 1D
Score: 6

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☑ Initializes and accumulates the frog's position at most `maxHops` times (*must be in context of a loop*)
3: ☑ Determines if a distance representing multiple hops is at least `goalDistance`
4: ☐ Does not determine if a distance representing multiple hops is less than starting position
5: ☐ Returns `false` if goal never reached. Does not return `false` if position ever less than starting position. Does not immediately return `true` if goal reached
6: ☐ Calls `simulate` more than the specified number of times
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 1E
Score: 5

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☐ Initializes and accumulates the frog's position more than `maxHops` times
3: ☑ Determines if a distance representing multiple hops is at least `goalDistance`
4: ☐ Does not determine if a distance representing multiple hops is less than starting position
5: ☐ Returns `false` if goal never reached. Does not return `false` if position ever less than starting position. Does not immediately return `true` if goal reached
6: ☐ Calls `simulate` an infinite number of times
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

Sample Identifier: 1F
Score: 4

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☐ Initializes the frog's position but does not accumulate the position within the context of a for loop
3: ☐ Does not determine if a distance representing multiple hops is at least `goalDistance`
4: ☐ Does not determine if a distance representing multiple hops is less than starting position
5: ☐ Does not return `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☐ Calls `simulate` incorrectly, and more than the specified number of times
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 1G
Score: 3

1: ☐ Incorrectly calls `hopDistance` to adjust the frog's position
2: ☐ Incorrectly implements the loop resulting in the frog's position not being accumulated correctly
3: ☐ Does not determine if a distance representing multiple hops is greater than `goalDistance`
4: ☐ Does not determine if a distance representing multiple hops is less than starting position
5: ☐ Does not return `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☐ Does not call `simulate` the specified number of times due to an incorrectly constructed loop header
7: ☑ Initializes and accumulates a count of `true` results
8: ☑ Calculates proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

Sample Identifier: 1H
Score: 2

1: ☑ Calls `hopDistance` and uses returned distance to adjust (or represent) the frog's position
2: ☐ Initializes the frog's position but does not accumulate the position within the context of a for loop
3: ☐ Does not determine if a distance representing multiple hops is at least `goalDistance`
4: ☐ Does not determine if a distance representing multiple hops is less than starting position
5: ☐ Does not return `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☐ Calls `simulate` more than the specified number of times
7: ☐ Does not initialize the variable before accumulating a count of `true` results
8: ☐ Incorrectly calculates proportion of successful simulations
9: ☑ Returns calculated value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 1I
Score: 1

1: ☐ Incorrectly calls `hopDistance` when attempting to represent the frog's position
2: ☐ Incorrectly initializes the variable used to accumulate the frog's position
3: ☐ Incorrectly determines if a distance representing multiple hops is at least `goalDistance`
4: ☐ Incorrectly determines if a distance representing multiple hops is less than starting position
5: ☐ Does not return `true` if goal ever reached, `false` if goal never reached or position ever less than starting position
6: ☐ Does not call `simulate` within the context of a loop
7: ☐ Does not initialize or accumulate a count of `true` results
8: ☐ Does not correctly calculate proportion of successful simulations using `double` arithmetic
9: ☑ Returns calculated value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

2. This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }

}
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects.

A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects `(words[i], words[j])` for every `i` and `j`, where $0 \le i < j <$ `words.length`. Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(b) Write the `WordPairList` method `numMatches`. This method returns the number of `WordPair` objects in `allPairs` for which the two strings match.

For example, the following code segment creates a `WordPairList` object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return `2`.

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete method `numMatches` below.

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 2: Word Pair

Part (a)

```
public WordPairList(String[] words)
{
   allPairs = new ArrayList<WordPair>();

   for (int i = 0; i < words.length-1; i++)
   {
      for (int j = i+1; j < words.length; j++)
      {
         allPairs.add(new WordPair(words[i], words[j]));
      }
   }
}
```

Part (b)

```
public int numMatches()
{
   int count = 0;

   for (WordPair pair: allPairs)
   {
      if (pair.getFirst().equals(pair.getSecond()))
      {
         count++;
      }
   }
   return count;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 2: Word Pair

Part (a)	`WordPairList`	**5 points**

Intent: *Form pairs of strings from an array and add to an* `ArrayList`

+1 Creates new `ArrayList` and assigns to `allPairs`

+1 Accesses all elements of `words` (*no bounds errors*)

+1 Constructs new `WordPair` using distinct elements of `words`

+1 Adds all necessary pairs of elements from word array to `allPairs`

+1 **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs

Part (b)	`numMatches`	**4 points**

Intent: *Count the number of pairs in an* `ArrayList` *that have the same value*

+1 Accesses all elements in `allPairs` (*no bounds errors*)

+1 Calls `getFirst` or `getSecond` on an element from list of pairs

+1 Compares first and second components of a pair in the list

+1 Counts number of matches of pair-like values

Question-Specific Penalties

-1 (z) Constructor returns a value

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 2: Scoring Notes

Part (a) `WordPairList`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Creates new `ArrayList` and assigns to `allPairs`	• `allPairs = new ArrayList();` • `allPairs = new ArrayList<>();` • `this.allPairs = ...`	• initialize a local variable that is never assigned to `allPairs`
+1	Accesses all elements of `words` (*no bounds errors*)		
+1	Constructs new `WordPair` using distinct elements of `words`		
+1	Adds all necessary pairs of elements from word array to `allPairs`	• have a loop bounds error • add unnecessary pairs	• improperly add to an `ArrayList`, e.g. `allPairs.get(i) = x;` • only add consecutive pairs `(words[i], words[i+1])`
+1	**On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs	• improperly add to an `ArrayList`, e.g. `allPairs.get(i) = x;` • have a loop bounds error	• add pairs (i, i) or (i, j) where i > j
Part (b) `numMatches`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Accesses all elements in `allPairs` (*no bounds errors*)		• access elements of `allPairs` as array elements (e.g. `allPairs[i]`)
+1	Calls `getFirst` or `getSecond` on an element from list of pairs		
+1	Compares first and second components of a pair in the list		• compare using `==`
+1	Counts number of matches of pair-like values		• fail to initialize the counter

Return is not assessed in part (b).

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators **(× • ÷ ≤ ≥ <> ≠)**
- `[]` vs. `()` vs. `<>`
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "*ArayList" instead of "ArrayList". As a counter example, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete the `WordPairList` constructor below.

2Aa

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words) {
    ArrayList<WordPair> p = new ArrayList<WordPair>();
    for (int i=0 ; i<(words.length-1); i++ ) {
        for (int j = i+1 ; j< words.length; j++) {
            p.add (new WordPair (words[i] , words[j] ));
        }
    }
    allPairs = p ;
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2Ab

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()  {
    int count = 0;

    for (int i=0; i< allPairs.size(); i++) {
        WordPair temp = allPairs.get(i);
        if((temp.getFirst()).equals(temp.getSecond())) {
            count ++;
        }
    }
    return count;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Ba

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
    allPairs = new ArrayList<WordPair>();
    int count=0;
    for(int i=0; i< words.length-1; i++){
        for(int j=i+1; j<words.length; j++){
            allPairs.get(count) = new WordPair(words[i], words[j]);
            count++;
        }
    }
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method numMatches below.

2Bb

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
        int count = 0;
        for(int i=0; i < allPairs.size(); i++){
            if(allPairs.get(i).getFirst().equals(allPairs.get(i).getSecond())){
                count++;
            }
        }
        return count;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Ea

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
{
ArrayList<WordPair> temp = new ArrayList(WordPair);

for (int i = 0; i < words.length-1; i++)
  {
    for (int y = i+1 ; y < words.length ; y++)
    {
      temp.add (words[i], words[y]);
    }
  }
allPairs = temp;
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method numMatches below.

2C6

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
{
  int cant = 0;
  for (int i = 0; i < allPairs.length; i ++)
  {  if (allPairs[i].getFirst().compareTo(allPairs[i].getSecond())
          == 0)
     {  cant ++;
     }
  }
  return cant;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Da

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
```

```
public WordPairList(String[] words)
{ for(int x=0; x<words.length-1; x++)
  { for(int y=x; y<words.length-1; y++)
    { if(y != x)
      { allPairs.add(new WordPair(words[x],words[y]);}}}}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2Db

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
```

```
public int numMatches()
{ int snail = 0;
  for(int i=0; i< allPairs.size(); i++)
  { if( allPairs.get(i).getFirst == allPairs.get(i).getSecond)
      {snail++;}}
  return snail; }
```

Unauthorized copying or reuse of any part of this page is illegal.

Come back and treat as str'g

* Complete the `WordPairList` constructor below.

2Ea

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words) {
```

```
for(int i=0; i < words.length-1; i++){
allPairs.add((new WordPair(words[i], words[i+1])));
}
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2Eb

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches() {
int matches = 0;
String one = "";
String two = "";
for(int i=0; i< allPairs.length(); i++){
    one = (allPairs[i].getFirst());
    two = (allPairs[i].getSecond());
    if (one.equals(two)) {
        matches += 1;
    }
}

return matches;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Fa

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
{
  for(int x=0; x < words.length(); x++
  {
    for(int y=x+1; y<= words.length(); y++)
    {
      allPairs.add(words[x], words[y]);
    }
  }
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2Fb

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
{
  int count=0;
  for(int x=0; x< allPairs.size(); x++)
  {
    if(allPairs.getFirst(x).equals(allPairs.getSecond(x)))
    {
      count++;
    }
  }
  return count;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Ga

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
{ int counter = 0;
  for (int w = 0; w < words.length-1; w++)
  {
    for (int p = 1; p < words.length-1; p++)
    {
      allPairs.set(counter, WordPair(phrase[w], phrase[p]));
      counter++;
    }
  }
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2 Gb

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
    { int count = 0;
      for (int i = 0; i < allPairs.size(); ++i)
        {
          if (allPairs.get(i).getFirst() == allPairs.get(i).getSecond())
            {
              count ++;
            }
        }
      return count;
    }
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Ha

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words) {
  WordPair allPairs = new ArrayList;
  for(int i=0; i< words.length; i++){
  allPairs.add( words[i]);
  }
}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `numMatches` below.

2H6

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
int count = 0;
for (int i = 0; i < words.length; i++) {
if ((WordPair) words[i].getFirst == ((WordPair) words[i].getSecond)) {
count++;
}
return count;
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete the `WordPairList` constructor below.

2Ia

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
```

```
for(int i = 0; i < words.size(); i++)
{
    int j = 1;

    String[] allPairs = {words[i], words[j]};

}
```

Part (b) begins on page 12

Unauthorized copying or reuse of any part of this page is illegal.

Complete method numMatches below.

2Ib

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
{
    for( int i = 0; i <= allPairs.size(); i++)
    {
        int j = 0;
        int k = 0;

        if( allPairs[j] == allPairs[k])
        {
            numMatches++;
            j++;
            k++;
        }

    }

    return numMatches;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Question 2

Sample Identifier: 2A
Score: 9

1: ☑ Creates new `ArrayList` and assigns to `allPairs`
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☑ Constructs new `WordPair` using distinct elements of `words`
4: ☑ Adds all necessary pairs of elements from word array to `allPairs`
5: ☑ **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs
6: ☑ Accesses all elements in `allPairs` (*no bounds errors*)
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☑ Compares first and second components of a pair in the list
9: ☑ Counts number of matches of pair-like values

Sample Identifier: 2B
Score: 8

1: ☑ Creates new `ArrayList` and assigns to `allPairs`
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☑ Constructs new `WordPair` using distinct elements of `words`
4: ☐ The response improperly adds all necessary pairs of elements to `allPairs` by assigning to a method call
5: ☑ **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs
6: ☑ Accesses all elements in `allPairs` (*no bounds errors*)
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☑ Compares first and second components of a pair in the list
9: ☑ Counts number of matches of pair-like values

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 2C
Score: 7

1: ☑ Creates new `ArrayList` and assigns to `allPairs`
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☐ The response is missing the construction of a new `WordPair` object
4: ☑ Adds all necessary pairs of elements from word array to `allPairs`
5: ☑ **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs
6: ☐ The response accesses elements of `allPairs` as array elements instead of list elements
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☑ Compares first and second components of a pair in the list
9: ☑ Counts number of matches of pair-like values

Sample Identifier: 2D
Score: 6

1: ☐ The response has no statement to create a new `ArrayList`
2: ☐ The response accesses all but the last element of `words`, causing a bounds error
3: ☑ Constructs new `WordPair` using distinct elements of `words`
4: ☑ Adds all necessary pairs of elements from word array to `allPairs`
5: ☑ **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs
6: ☑ Accesses all elements in `allPairs` (*no bounds errors*)
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☐ The response incorrectly compares the components of a pair in the list by using the `==` operator
9: ☑ Counts number of matches of pair-like values

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 2E
Score: 5

1: ☐ The response has no statement to create a new `ArrayList`
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☑ Constructs new `WordPair` using distinct elements of `words`
4: ☐ The response only adds consecutive pairs of elements from word array to `allPairs`
5: ☐ **On exit:** `allPairs` contains only consecutive pairs instead of all necessary pairs
6: ☐ The response accesses elements of `allPairs` as array elements instead of list elements
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☑ Compares first and second components of a pair in the list
9: ☑ Counts number of matches of pair-like values

Sample Identifier: 2F
Score: 4

1: ☐ The response has no statement to create a new `ArrayList`
2: ☐ The response traverses past the end of the `words` array when accessing elements in the inner loop, causing a bounds error
3: ☐ The response is missing the construction of a new `WordPair` object
4: ☑ Adds all necessary pairs of elements from word array to `allPairs`
5: ☑ **On exit:** `allPairs` contains all necessary pairs and no unnecessary pairs
6: ☐ The response is missing the call to the `get` method to access the list
7: ☐ The response incorrectly passes the index as a parameter to both the `getFirst` and `getSecond` methods
8: ☑ Compares first and second components of a pair in the list
9: ☑ Counts number of matches of pair-like values

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 2G
Score: 3

1: ☐ The response has no statement to create a new `ArrayList`
2: ☐ The response accesses the non-existent array `phrase` and does not access any elements of the array `words`
3: ☐ The response is missing `new` in the construction of a `WordPair`
4: ☐ The response improperly adds all necessary pairs of elements to `allPairs` by using the `ArrayList` method `set`
5: ☐ **On exit:** `allPairs` contains all possible pairs so it contains many unnecessary pairs
6: ☑ Accesses all elements in `allPairs` (*no bounds errors*)
7: ☑ Calls `getFirst` or `getSecond` on an element from list of pairs
8: ☐ The response incorrectly compares the components of a pair in the list by using the `==` operator
9: ☑ Counts number of matches of pair-like values

Sample Identifier: 2H
Score: 2

1: ☐ The response creates and assigns a new `ArrayList` to a local variable named `allPairs` instead of the required instance variable
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☐ The response is missing the construction of a new `WordPair` object
4: ☐ The response does not add any pairs of elements to `allPairs`
5: ☐ **On exit:** `allPairs` does not contain any pairs of elements
6: ☐ Accesses all elements of the unavailable `words` array instead of `allPairs`
7: ☐ The response does not call the methods `getFirst` or `getSecond` using an element from a list of pairs
8: ☐ The response incorrectly compares the components of a pair by using the `==` operator and the pair is not from a list
9: ☑ Counts number of matches of pair-like values

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 2I
Score: 1

1: ☐ The response has no statement to create a new `ArrayList`
2: ☑ Accesses all elements of `words` (*no bounds errors*)
3: ☐ The response is missing the construction of a new `WordPair` object
4: ☐ The response does not add any pairs of elements to `allPairs`
5: ☐ **On exit:** `allPairs` does not contain any pairs of element
6: ☐ The response accesses elements of `allPairs` as array elements instead of list elements
7: ☐ The response has no calls to the `getFirst` or `getSecond` methods
8: ☐ The response compares `WordPair` objects instead of components of the word pair and incorrectly uses the `==` operator for the comparison
9: ☐ The response does not declare or initialize the accumulator for the number of matches of pair-like values

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

3. The `StringChecker` interface describes classes that check if strings are valid, according to some criterion.

```
public interface StringChecker
{
    /** Returns true if str is valid. */
    boolean isValid(String str);
}
```

A `CodeWordChecker` is a `StringChecker`. A `CodeWordChecker` object can be constructed with three parameters: two integers and a string. The first two parameters specify the minimum and maximum code word lengths, respectively, and the third parameter specifies a string that must not occur in the code word. A `CodeWordChecker` object can also be constructed with a single parameter that specifies a string that must not occur in the code word; in this case the minimum and maximum lengths will default to 6 and 20, respectively.

The following examples illustrate the behavior of `CodeWordChecker` objects.

Example 1

```
StringChecker sc1 = new CodeWordChecker(5, 8, "$");
```

Valid code words have 5 to 8 characters and must not include the string `"$"`.

Method call	Return value	Explanation
`sc1.isValid("happy")`	`true`	The code word is valid.
`sc1.isValid("happy$")`	`false`	The code word contains `"$"`.
`sc1.isValid("Code")`	`false`	The code word is too short.
`sc1.isValid("happyCode")`	`false`	The code word is too long.

Example 2

```
StringChecker sc2 = new CodeWordChecker("pass");
```

Valid code words must not include the string `"pass"`. Because the bounds are not specified, the length bounds are 6 and 20, inclusive.

Method call	Return value	Explanation
`sc2.isValid("MyPass")`	`true`	The code word is valid.
`sc2.isValid("Mypassport")`	`false`	The code word contains `"pass"`.
`sc2.isValid("happy")`	`false`	The code word is too short.
`sc2.isValid("1,000,000,000,000,000")`	`false`	The code word is too long.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 3: Code Word Checker

```
public class CodeWordChecker implements StringChecker
{
   private int minLength;
   private int maxLength;
   private String notAllowed;

   public CodeWordChecker (int minLen, int maxLen, String symbol)
   {
      minLength = minLen;
      maxLength = maxLen;
      notAllowed = symbol;
   }

   public CodeWordChecker (String symbol)
   {
      minLength = 6;
      maxLength = 20;
      notAllowed = symbol;
   }

   public boolean isValid(String str)
   {
      return str.length() >= minLength && str.length() <= maxLength &&
             str.indexOf(notAllowed) == -1;
   }

}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 3: Code Word Checker

Class: `CodeWordChecker` **9 points**

Intent: *Define implementation of a class to determine if a string meets a set of criteria*

+1 Declares header: `public class CodeWordChecker implements StringChecker`

+1 Declares all appropriate `private` instance variables

+3 Constructors

- **+1** Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
- **+1** Uses all parameters to initialize instance variables in 3-parameter constructor
- **+1** Uses parameter and default values to initialize instance variables in 1-parameter constructor

+4 `isValid` method

- **+1** Declares header: `public boolean isValid(String __)`
- **+1** Checks for length between min and max inclusive
- **+1** Checks for unwanted string
- **+1** Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise

Question-Specific Penalties

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0
Question 3: Scoring Notes

Class `CodeWordChecker`			**9 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Declares header: `public class CodeWordChecker implements StringChecker`	• omit keyword `public`	• declare class `private` • declare class `static`
+1	Declares all appropriate `private` instance variables		• declare variables as `static` • omit keyword `private` • declare variables outside the class
+3	Constructors		
+1	Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`	• omit keyword `public`	• declare method `static` • declare method `private`
+1	Uses all parameters to initialize instance variables in 3-parameter constructor		• fail to declare instance variables • initialize local variables instead of instance variables • assign variables to parameters
+1	Uses parameter and default values to initialize instance variables in 1-parameter constructor	• initialize instance variables to default values when declared	• fail to declare instance variables • initialize local variables instead of instance variables • assign variables to parameters
+4	`isValid` method		
+1	Declares header: `public boolean isValid (String __)`		• fail to declare method `public` • declare method `static`
+1	Checks for length between min and max inclusive		• fail to use instance variables • fail to declare the method header
+1	Checks for unwanted string		• fail to use instance variables • fail to declare the method header
+1	Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise	• have incorrect checks for length and/or containment, but return the correct value based on those checks	• fail to declare the method header • fail to return in all cases • only check one substring location for containment

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators **(× • ÷ ≤ ≥ <> ≠)**
- `[]` vs. `()` vs. `<>`
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "*ArayList" instead of "ArrayList". As a counter example, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

3A

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

```
public class CodeWordChecker implements StringChecker
{
    private int min = 6;
    private int max = 20;
    private String badWord;
    public CodeWordChecker(int min, int max, String badWord)
    {
        this.min = min;
        this.max= max;
        this.badWord = badWord;
    }
    public CodeWordChecker(String badWord)
    {
        this.badWord = badWord;
    }
    public boolean isValid(String str)
    {
        if(str.length()<min || str.length()>max || str.indexOf(badWord)!=-1)
        {
            return fake;
        }
        else
        {
            return true;
        }
    }
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

3B

```
public class CodeWordChecker implements StringChecker
{
   private int min, max;
   private String forbidden;
   public CodeWordChecker(int mi, int ma, String f)
   {
      min = mi;
      max = ma;
      forbidden = f;
   }
   public CodeWordChecker(String f)
   {
      min = b;
      max = 2b;
      forbidden = f;
   }
   public boolean isValid(String str)
   {
      if(str.length() < min || str.length() > max)
         return false;
      for(int i=0; i < str.length() - forbidden.length(); i++)
      {
         if (str.substring(i, i + forbidden.length()).equals(forbidden)
            return false;
      }
      return true;
   }
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3C

```
Public class CodeWordChecker implements StringChecker
{
   Private int minimum
   Private int maximum
   Private String notOccur;

   public CodeWordChecker(int min, int max, String str)
   {
     minimum = min;
     maximum = max;
     notOccur = str;
   }
   public CodeWordChecker(String str)
   {
     minimum = 6;
     maximum = 20;
     notOccur = str;
   }

   public boolean isValid(String str)
   {
      if(str.length() >= minimum && str.length() <= maximum)
      {
         for(int i=0 ; i < str.length(); i++)
         {
            if(!str.substring(i, i+notOccur.length()).equals(notOccur))
               return true;
            else
               return false;
         }
      }
      return false;
   }
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3D

```
public class CodeWordChecker implements StringChecker
{
  int min;
  int max;
  String s;

    public CodeWordChecker(int a, int b, String c)
    {
     min = a;
     max = b;
     s = c;
    }
    public CodeWordChecker (string d)
    {
      s = d;
     min = 6;
     max = 20;
    }
    public boolean isValid(String str)
    {
      if (str.length() >= min && str.length <= max)
          if (s.indexOf(str) == -1)
            return true;
          else
            return false;

    }
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3E

```
Public class CodeWordChecker implements StringChecker{
Public int lowerBound, upperBound;
Public string ext;
    Public class CodeWordChecker (string und){
        ext= und;
        lowerBound = 6;
        upperBound =20;
    }
Public Class CodeWordChecker(int str, int fin, string and){
        lowerBound = str;
        upperBound = fin;
        ext= und;
    }

Public boolean isValid(string sta){
   if( sta.length>=upperBound || sta.length<= lowerBound)
            return false;
   if( sta.indexOf(ext) > 0){
            return false;
    }
    return true;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3F

```
class CodeWordChecker implements StringChecker {
    int low;
    int high;
    String noGo;

    public CodeWordChecker (String oneInput) {
        low = 6;
        high = 20;
        noGo = oneInput;
    }
    public CodeWordChecker (int l, int h, String thirdInput) {
        low = l;
        high = h;
        noGo = thirdInput;
    }
}
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3G

```
public class CodeWordChecker implements StringChecker {

    public CodeWordChecker(int min; int max, String word) {
        int minLength = min;
        int maxLengt = max;
        String excludedWord=word;
    }

    public CodeWordChecker(String word){
        int minLength=6;
        int maxLength=20;
        String excludedWord = word;
    }

    public Boolean isValid(String word){
        if(word.compareTo(excludedWord)=0){
            return false;
        }else if(word.length()<minLength || word.length()>maxLength){
            return false;
        }else{
            return true;
        }

}
```

Unauthorized copying or reuse of any part of this page is illegal.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3H

```
int min=0;
int max=0;
String invalid = null;

Public CodeWord Checker (int a; int b, String s) {
    min = a;
    max = b;
    s = invalid;
}

Public Boolean isValid (String s){
    if (s.length > max)
        return False;
    if (s.length < min)
        return False;

    For (x=0; x< s.length; x++) {
        String sub = s.Substring(x);
        If (sub.equals(s))
        return False;
    }
    Else return true;
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples. 3I

```
public class codeWordChecker(int min, int max,
                             string valid)
{

    public boolean isValid(String str)
    {
        if (str.length() > min && str.length() < max)
        {
            if (str.contains(valid) == -1)
            {
                return false;
            }
        }
        else
        {
        return false;
        }
    }
}
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Question 3

Sample Identifier: 3A
Score: 9

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☑ Declares all appropriate `private` instance variables
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☑ Checks for length between min and max inclusive
8: ☑ Checks for unwanted string
9: ☑ Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise

Sample Identifier: 3B
Score: 8

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☑ Declares all appropriate `private` instance variables
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☑ Checks for length between min and max inclusive
8: ☐ Failed to examine all substrings. Loop exits early (off by one).
9: ☑ Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 3C
Score: 7

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☑ Declares all appropriate `private` instance variables
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☑ Checks for length between min and max inclusive
8: ☐ Loop will result in an index out of bounds error
9: ☐ An early return prevents examining all cases

Sample Identifier: 3D
Score: 6

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☐ Did not declare instance variables `private`
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☑ Checks for length between min and max inclusive
8: ☐ Incorrect check for unwanted string. String identifiers are reversed.
9: ☐ Fails to return `false` when length is not between min and max

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 3E
Score: 5

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☐ Did not declare instance variables `private`
3: ☐ Included `class` in header
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☐ The `false` return should not include max and min
8: ☐ Failed to include `= 0` when returning `false`
9: ☑ Returns `true` if length is between min and max and does not contain the unwanted string, `false` otherwise

Sample Identifier: 3F
Score: 4

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☐ Failed to declare instance variables `private`
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☑ Uses all parameters to initialize instance variables in 3-parameter constructor
5: ☑ Uses parameter and default values to initialize instance variables in 1-parameter constructor
6: ☐ Failed to declare header: `public boolean isValid(String __)`
7: ☐ Failed to implement method `isValid`
8: ☐ Failed to implement method `isValid`
9: ☐ Failed to implement method `isValid`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 3G
Score: 3

1: ☑ Declares header: `public class CodeWordChecker implements StringChecker`
2: ☐ Did not declare `private` instance variables
3: ☑ Declares headers: `public CodeWordChecker(int __, int __, String __)` and `public CodeWordChecker(String __)`
4: ☐ Failed to declare instance variables
5: ☐ Failed to declare instance variables
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☐ Failed to declare instance variables
8: ☐ Failed to declare instance variables
9: ☐ Failed to check all cases; only identified one occurrence

Sample Identifier: 3H
Score: 2

1: ☐ Failed to declare header
2: ☐ Failed to declare instance variables `private`
3: ☐ Failed to declare 1-parameter constructor
4: ☐ The instance variable has been assigned to the parameter
5: ☐ Failed to implement 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☑ Checks for length between min and max inclusive
8: ☐ Failed to use the number of characters in the unwanted string
9: ☐ Failed to properly return `true` due to dangling else

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 3I
Score: 1

1: ☐ Failed to declare header properly
2: ☐ Failed to declare instance variables
3: ☐ Failed to declare constructor headers
4: ☐ Failed to implement 3-parameter constructor
5: ☐ Failed to implement 1-parameter constructor
6: ☑ Declares header: `public boolean isValid(String __)`
7: ☐ Failed to declare instance variables
8: ☐ Failed to declare instance variables
9: ☐ Failed to return `true`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

4. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {  /* implementation not shown */  }


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {  /* to be implemented in part (b) */  }

}
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row `r` and column `c`.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable `result` will have the following contents.

```
result: {1, 4, 7, 5}
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

(b) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a *Latin square*, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The `ArrayTester` class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array `arr` contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in `arr1` appears in `arr2`. You do not need to write the code for these methods.

Class information for this question

```
public class ArrayTester

public static int[] getColumn(int[][] arr2D, int c)
public static boolean hasAllValues(int[] arr1, int[] arr2)
public static boolean containsDuplicates(int[] arr)
public static boolean isLatin(int[][] square)
```

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

STOP

END OF EXAM

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 4: Latin Squares

Part (a)

```
public static int[] getColumn(int[][] arr2D, int c)
{
   int [] result = new int[arr2D.length];

   for (int r = 0; r < arr2D.length; r++)
   {
      result[r] = arr2D[r][c];
   }
   return result;
}
```

Part (b)

```
public static boolean isLatin(int[][] square)
{
   if (containsDuplicates(square[0]))
   {
      return false;
   }

   for (int r = 1; r < square.length; r++)
   {
      if (!hasAllValues(square[0], square[r]))
      {
         return false;
      }
   }

   for (int c = 0; c < square[0].length; c++)
   {
      if (!hasAllValues(square[0], getColumn(square,c))
      {
         return false;
      }
   }

   return true;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 4: Latin Squares

Part (a) `getColumn` **4 points**

Intent: *Create a 1-D array that contains the values from one column of a 2-D array*

+1 Constructs a new `int` array of size `arr2D.length`

+1 Accesses all items in one column of `arr2D` (*no bounds errors*)

+1 Assigns one element from `arr2D` to the corresponding element in the new array

+1 **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order

Part (b) `isLatin` **5 points**

Intent: *Check conditions to determine if a square 2-D array is a Latin square*

+1 Calls `containsDuplicates` referencing a row or column of `square`

+1 Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column

+1 Applies `hasAllValues` to all rows or all columns (*no bounds errors*)

+1 Calls `getColumn` to obtain a valid column from `square`

+1 Returns `true` if all three Latin square conditions are satisfied, `false` otherwise

Question-Specific Penalties

-1 (r) incorrect construction of a copy of a row

-1 (s) syntactically incorrect method call to any of `getColumn()`, `containsDuplicates()`, or `hasAllValues()`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Question 4: Scoring Notes

Part (a) `getColumn`			**4 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Constructs a new `int` array of size `arr2D.length`		• only create an `ArrayList`
+1	Accesses all items in one column of `arr2D` (*no bounds errors*)	• declare the new array of an incorrect size and use that size as the number of loop iterations	• switch row and column indices
+1	Assigns one element from `arr2D` to the corresponding element in the new array		• use `ArrayList` methods to add to array
+1	**On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order		• switch row and column indices • do not use an index when assigning values to the array
Part (b) `isLatin`			**5 points**
Points	Rubric Criteria	Responses earn the point if they...	Responses will not earn the point if they...
+1	Calls `containsDuplicates` referencing a row or column of `square`	• reference any row or column of `square`, even if the syntax of the reference is incorrect	
+1	Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column	• reference any two distinct rows, two distinct columns, or a row and column of `square`, even if the syntax of the reference is incorrect.	
+1	Applies `hasAllValues` to all rows or all columns (*no bounds errors*)		• only reference one array in the call to `hasAllValues`
+1	Calls `getColumn` to obtain a valid column from `square`		• reverse parameters
+1	Returns `true` if all three Latin square conditions are satisfied, `false` otherwise	• test the three sets of conditions and return the correct value	

Return is not assessed in Part (a).

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® COMPUTER SCIENCE A
2018 SCORING GUIDELINES — Version 1.0

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators **(× • ÷ ≤ ≥ <> ≠)**
- `[]` vs. `()` vs. `<>`
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "*ArayList" instead of "ArrayList". As a counter example, note that if the code declares "`int G=99, g=0;`", then uses "`while (G < 10)`" instead of "`while (g < 10)`", the context does **not** allow for the reader to assume the use of the lower case variable.*

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

4Aa

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int colSize = arr2D.length;
    int[] column = new int[colSize];

    for (int r=0; r<colSize; r++)
    {
        column[r] = arr2D[r][c];
    }

    return column;
}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

4Ab

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    int[] firstRow = square[0];
    if (containsDuplicates(firstRow))
        return false;

    for (int x=1 ; x < square.length; x++)
    {
      int[] temp = square[x];
      if (!hasAllValues(firstRow, temp))
          return false;
    }

    for (int x=1; x < square.length; x++)
    {
      int[] temp = getColumn(square, x);
      if (!hasAllValues(firstRow, temp))
          return false;
    }

    return true;

}
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method `getColumn` below.

4Ba

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int[] col = new int[arr2D.length];

    for(int i=0; i<arr2D.length; i++){
        col[i] = arr2D[i][c];
    }

    return col;

}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit. 4B6

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

```
for(int i = 0; i < square.length; i++){
    if (!(hasAllValues(squares[0], squares[i]))) {
        return false;
    }
}
for (int i = 0; i < squares[0].length; i++) {
    if (!(hasAllValues(squares[0], getColumn(squares, i)))
    {
     return false;
    }
}
return containsDuplicates(squares[0]);
}
```

Unauthorized copying or reuse of any part of this page is illegal.

4Ca

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c) {
    int [] result = new array (arr2D.length);

    for (int r=0; r < arr2D.length; r++) {

        result [r] = arr2D [r][c];

    }

    return result;

}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

4Ch

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    if ( containDuplicates (square[0]) )
        return false;

    for (int c=0; c < square[0].length-1; c++) {
        if(!(hasAllValues (getColumn(square,c), getColumn(square,c+1))))
        return false;
    }
    return true;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method `getColumn` below.

4Da

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int [] column = new int[arr2D[0].length];
    for(int i = 0; i < arr2D.length; i++)
    {
        int[i] = arr2D[i][c];
    }
    return column;
}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

4Db

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    if (square[0].containsDuplicates() == true)
    {
        return false;
    }

    for (int i = 1; i < square.length; i++)
    {
        if (hasAllValues(square[0], square[i]) == false)
        {
            return false;
        }
    }

    for (int c = 0; c < square[0].length; i++)
    {
        if (hasAllValues(square[0], getColumn(square, c)) == false)
        {
            return false;
        }
    }

    return true;
}
```

Unauthorized copying or reuse of any part of this page is illegal.

4Ea

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

```
int[] result = [array2D[c].length];
    for (int i == 0; i < array2D[c].length; i++){
        result[i] = array2D[i][c]; }

return result;
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

4Eb

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

```
if(containsDuplicates(square[0]) == true){
   returns false;}
boolean flag = false;
for(int i = 0 ; i < square[0].length ; i++){
   if (hasAllValues(square[i], square[i+1]) == true){
      flag = true;}}
return flag;
```

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method `getColumn` below.

0 1 2 3 4 5 4
0 1 2 3 4

4Fa

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

row, column

```
int[] column = new int [arr2D.length()];

for (x = arr2D.length(); x > 0 ... ; x--)

{ column[x] = arr2D[x, c] ;

}

return column;
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

4Fb

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

~~for (int x = 0; x > array~~

if ! square.getColumn = square[1]

return false

else if square[1].containsDuplicates

return false;

else if ! square.hasallvalues (square[1], square[2])

return false;

else

return true;

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `getColumn` below.

4Ga

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

```
int[] arrNew = new Arraylist[ arr2D.length.length];

for (i=0; i < arr2D.length ; i++){
        arrNew[i] = arr2D[i][c];
}

return arrNew;
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

GO ON TO THE NEXT PAGE.

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

46b

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

```
int[] arr1 = square[0];

if (!containsDuplicates) {
    for (i=0; i < square.length; i++) {
        int[] arr2 = square[i];
        if (hasAllValues) {
            for (j=0; j < square.length.length; j++) {
                int[] arr3 = new ArrayList[square.length];
                for (z=0; z < sqrr.length; z++) {
                    arr3.add(square[j][z]);
                }
                arr3 = arr2;
                if (hasAllValues) {
                    return True;
                }
            }
        }
    }
}
return false;
```

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `getColumn` below.

4Ha

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{ int[] arr = new int[arr2D[0][0].length()]
  for(int r=0; r<= 0; r++)
    for(int j=c; j< arr2D.length[0][0]; j++)
      int[arr[c]] = arr2D[r][c];

  return arr;

}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit. 4 Hb

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    if (containsDuplicates(square) = false)
      return false
    int[] arr = getColumn(square, 1)
    int[] arr2 = getColumn(square, 2)
    if(hasAllValues(arr, arr2) = true)
      return false
      else
      return true

}
```

Unauthorized copying or reuse of any part of this page is illegal.

4 Ia

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
{
    int[] newArr = new int[ ];
    for (int r=0; r<arr2D.length; r++)
    {
        if (arr2D[r][ ]=arr2D[r][c])
        newArr.add(arr2D[r][c]);
    }
    return newArr;
}
```

Part (b) begins on page 20

Unauthorized copying or reuse of any part of this page is illegal.

4Ib

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
{
    Array arr1 = new Array[ ];
    Array arr2 = new Array[ ];
    arr1 = square.length;
    arr2 = square.getColumn(square, square.length);
    for(int r=0; r< square.length; r++)
    {
        if(square[r][ ].containsDuplicates(arr1))
            latin = false;
        else
            for(int c=0; c< square[r].length; c++)
            {
                if(square[r][c].hasAllValues(arr1, arr2))
                    latin = true;
            }
    }
    return latin;
}
```

boolean latin = false;

Unauthorized copying or reuse of any part of this page is illegal.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Question 4

Sample Identifier: 4A
Score: 9

1: ☑ Constructs a new `int` array of size `arr2D.length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☑ Applies `hasAllValues` to all rows or all columns (*no bounds errors*)
8: ☑ Calls `getColumn` to obtain a valid column from `square`
9: ☑ Returns `true` if all three Latin square conditions are satisfied, `false` otherwise

Sample Identifier: 4B
Score: 8

1: ☑ Constructs a new `int` array of size `arr2D.length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☑ Applies `hasAllValues` to all rows or all columns (*no bounds errors*)
8: ☑ Calls `getColumn` to obtain a valid column from `square`
9: ☐ Returns `true` if all three Latin square conditions are satisfied, `false` otherwise. The third Latin square condition, which requires all rows and columns to be duplicate free, fails, because `containsDuplicates(squares[0])` is returned instead of `!containsDuplicates(squares[0])`.

**Note: The student response uses `squares` instead of `square` throughout part (b). This is a non-penalized spelling error.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 4C
Score: 7

1: ☐ The `int` array was incorrectly instantiated: `new array(arr2D.length)`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☑ Applies `hasAllValues` to all rows or all columns (*no bounds errors*)
8: ☑ Calls `getColumn` to obtain a valid column from `square`
9: ☐ Does not check the Latin square condition that every row of `square` must contain the same values

Sample Identifier: 4D
Score: 6

1: ☐ Constructs a new `int` array of incorrect size `arr2D[0].length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☐ The data type `int` is used instead of the array name `column`, when assigning a corresponding element of column `c` from `square`
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☑ Applies `hasAllValues` to all rows or all columns (*no bounds errors*)
8: ☑ Calls `getColumn` to obtain a valid column from `square`
9: ☑ Returns `true` if all three Latin square conditions are satisfied, `false` otherwise

-1 (s) syntactically incorrect method call to any of `getColumn()`, `containsDuplicates()`, or `hasAllValues()`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 4E
Score: 5

1: ☐ Constructs a new `int` array of incorrect size `arr2D[0].length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☐ There is a boundary error when applying `hasAllValues` to all rows
8: ☐ There is no call to the method `getColumn`.
9: ☐ Does not check the Latin square condition that every column of `square` must contain the same values

Sample Identifier: 4F
Score: 4

1: ☑ Constructs a new `int` array of size `arr2D.length`
2: ☐ There is a boundary error when accessing all items in one column of `arr2D`
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☑ Calls `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☐ Does not apply `hasAllValues` to all rows or all columns
8: ☐ There is no call to the method `getColumn`
9: ☐ Does not check all Latin square conditions

-1 (s) syntactically incorrect method call to any of `getColumn()`, `containsDuplicates()`, or `hasAllValues()`

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 4G
Score: 3

1: ☐ Incorrectly constructs a new `int` array of size `arr2D.length.length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☑ Assigns one element from `arr2D` to the corresponding element in the new array
4: ☑ **On exit:** The new array has all the elements from the specified column in `arr2D` in the correct order
5: ☐ Does not call `containsDuplicates` referencing a row or column of `square`
6: ☐ Does not call `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☐ Does not apply `hasAllValues` to all rows or all columns (*no bounds errors*)
8: ☐ Does not call `getColumn` to obtain a valid column from `square`
9: ☐ Does not check any Latin square conditions

Sample Identifier: 4H
Score: 2

1: ☐ Incorrectly constructs a new `int` array of size `arr2D[0][0].length`
2: ☐ Does not access any items in one column of `arr2D`. The inner loop does not execute.
3: ☐ Does not assign one element from `arr2D` to the corresponding element in the new array. Redeclares the array as `int arr`.
4: ☐ **On exit:** The new array does not have all the elements from the specified column in `arr2D` in the correct order
5: ☐ Does not call `containsDuplicates` referencing a row or column of `square`
6: ☑ Calls `hasAllValues` referencing two different rows, two different columns, or one row and one column
7: ☐ Does not apply `hasAllValues` to all rows or all columns. No loop exists.
8: ☑ Calls `getColumn` to obtain a valid column from `square`
9: ☐ Does not check any Latin square conditions.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.

AP® SAMPLE STUDENT RESPONSES AND SCORING NOTES
2018 AP COMPUTER SCIENCE A

Sample Identifier: 4I
Score: 1

1: ☐ Does not construct a new `int` array of size `arr2D.length`
2: ☑ Accesses all items in one column of `arr2D` (*no bounds errors*)
3: ☐ Attempts to assign one element from `arr2D` to the corresponding element in the new array using the method `add`
4: ☐ **On exit:** The new array does not have all the elements from the specified column in `arr2D` in the correct order
5: ☐ Does not call `containsDuplicates` referencing a row or column of `square`. The array `arr1` does not contain any values of `square`.
6: ☐ Does not call `hasAllValues` referencing two different rows, two different columns, or one row and one column. The arrays `arr1` and `arr2` do not contain any values of `square`.
7: ☐ Does not apply `hasAllValues` to all rows or all columns. The arrays `arr1` and `arr2` do not contain any values of `square`.
8: ☐ Does not obtain a valid column from `square` when `getColumn` is called.
9: ☐ Does not check any Latin square conditions.

© 2018 The College Board.
Visit the College Board on the Web: www.collegeboard.org.